Turbo IP Core

User Guide

Subscribe

Send Feedback

UG-TURBO
2015.11.11

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Contents

About the Turbo IP Core 1

2015.11.11

UG-TURBO Subscribe Send Feedback

Related Information

- **Introduction to Altera IP Cores**
  Provides general information about all Altera IP cores, including parameterizing, generating, upgrading, and simulating IP.
- **Creating Version-Independent IP and Qsys Simulation Scripts**
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- **Project Management Best Practices**
  Guidelines for efficient management and portability of your project and IP files.

Altera DSP IP Core Features

- Avalon® Streaming (Avalon-ST) interfaces
- DSP Builder ready
- Testbenches to verify the IP core
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators

Turbo IP Core Features

- 3GPP LTE compliant.
- 3GPP UMTS compliant with support for block sizes from 40 to 5,114.
- C/MATLAB bit-accurate models for performance simulation or RTL test vector generation.

Decoder Features

- Successive interface cancellation (SIC) for the LTE-A channel coding enhancement over LTE.
- Run time parameters for interleaver size and number of iterations.
- Early termination with cyclical redundancy check (CRC).
- Compile time parameters for the number of parallel engines, choice of decoding algorithm, input precision, and output size.
- Double-buffering for reduced latency real-time applications, which allows the decoder to receive data while processing the previous data block.
- No external memory required.

Encoder Features

- 3GPP LTE and LTE-A compliant.
- 3GPP UMTS compliant with support for block sizes from 40 to 5,114.
- Run-time selectable interleaver block sizes .
- Code rate 1/3 only.
- Use external rate matching for other code rates.
- Double-buffering allows the encoder to receive data while processing the previous data block.

DSP IP Core Device Family Support

Altera® offers the following device support levels for Altera IP cores:

- Preliminary support—Altera verifies the IP core with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. You can use it in production designs with caution.
- Final support—Altera verifies the IP core with final timing models for this device family. The IP core meets all functional and timing requirements for the device family. You can use it in production designs.

Table 1-1: DSP IP Core Device Family Support

Device Family	Support
Arria® II GX	Final
Arria II GZ	Final
Arria V	Final
Arria 10	Final
Cyclone® IV	Final
Cyclone V	Final
MAX® 10 FPGA	Final
Stratix® IV GT	Final
Stratix IV GX/E	Final
Stratix V	Final

Device Family	Support
Other device families	No support

Turbo IP Core Release Information

Use the release information when licensing the IP core.

Table 1-2: Release Information

Item	Description
Version	15.1
Release Date	November 2015
Ordering Code	IP-TURBO (IPR-TURBO)
Product ID	00FC
Vendor ID	6AF7

Altera verifies that the current version of the Quartus Prime software compiles the previous version of each IP core. Altera does not verify that the Quartus Prime software compiles IP core versions older than the previous version. The *Altera IP Release Notes* lists any exceptions.

Related Information

- **Altera IP Release Notes**
- **Errata for Turbo IP core in the Knowledge Base**

DSP IP Core Verification

Before releasing a version of an IP core, Altera runs comprehensive regression tests to verify its quality and correctness. Altera generates custom variations of the IP core to exercise the various parameter options and thoroughly simulates the resulting simulation models with the results verified against master simulation models.

Turbo IP Core Performance and Resource Utilization

Table 1-3: Performance and Resource Utilization

Typical expected performance for a Turbo IP Core using the Quartus Prime software with the Arria V (5AGXFB3H4F35C5), Cyclone V (5CGXFC7C7F23C8), and Stratix V (5SGXEA7H3F35C3) devices.

Device	Parameters				ALM	Memory		fMAX (MHz)
	Codec Type	Standard	Input Bits	Engines		M20K	M10K	
Arria V	Encoder	LTE			434	-	2	237

Device	Parameters				ALM	Memory		fMAX (MHz)
	Codec Type	Standard	Input Bits	Engines		M20K	M10K	
Cyclone V	Encoder	LTE			435	-	2	175
Stratix V	Encoder	LTE			430	2	-	344
Arria V	Encoder	UMTS			959	-	4	151
Cyclone V	Encoder	UMTS			961	-	4	103
Stratix V	Encoder	UMTS			954	4	-	245
Arria V	Decoder	LTE	4	2	4,497	--	40	167
Cyclone V	Decoder	LTE	4	2	4,505	--	40	121
Stratix V	Decoder	LTE	4	2	4,138	27	-	251
Arria V	Decoder	LTE	4	4	6,194	--	37	175
Cyclone V	Decoder	LTE	4	4	6,221	--	37	132
Stratix V	Decoder	LTE	4	4	5,774	21	-	273
Arria V	Decoder	LTE	4	8	9,893	--	37	179
Cyclone V	Decoder	LTE	4	8	9,881	--	37	130
Stratix V	Decoder	LTE	4	8	9,049	21	-	253
Arria V	Decoder	LTE	8	2	5,998	--	58	156
Cyclone V	Decoder	LTE	8	2	6,001	--	58	122
Stratix V	Decoder	LTE	8	2	5,370	39	-	251
Arria V	Decoder	LTE	8	4	8,482	--	55	168
Cyclone V	Decoder	LTE	8	4	8,520	--	55	124
Stratix V	Decoder	LTE	8	4	7,658	30	--	256
Arria V	Decoder	LTE	8	8	13,672	--	54	161
Cyclone V	Decoder	LTE	8	8	13,690	--		117
Stratix V	Decoder	LTE	8	8	12,246	30	--	250
Arria V	Decoder	UMTS	4	2	3,935	--	81	133
Stratix V	Decoder	UMTS	4	2	3,622	71	--	217
Arria V	Decoder	UMTS	4	4	6,161	--	83	137
Cyclone V	Decoder	UMTS	4	4	6,141	--	83	100
Stratix V	Decoder	UMTS	4	4	5,661	73	--	215
Arria V	Decoder	UMTS	8	2	5,394	--	91	124
Cyclone V	Decoder	UMTS	8	2	5,390	--	91	93
Stratix V	Decoder	UMTS	8	2	4,646	81	--	194
Arria V	Decoder	UMTS	8	4	8,189	--	93	125

Device	Parameters				ALM	Memory		fMAX (MHz)
	Codec Type	Standard	Input Bits	Engines		M20K	M10K	
Cyclone V	Decoder	UMTS	8	4	8,190	--	93	92
Stratix V	Decoder	UMTS	8	4	7,381	78	--	188

Turbo Code Licensing Disclaimer

France Telecom, for itself and certain other parties, claims certain intellectual property rights covering Turbo Codes technology, and has decided to license these rights under a licensing program called the Turbo Codes Licensing Program. Supply of this IP core does not convey a license nor imply any right to use any Turbo Codes patents owned by France Telecom, TDF or GET. For information about the Turbo Codes Licensing Program, contact France Telecom at the following address:

France Telecom R&D

VAT/TURBOCODES

38, rue du Général Leclerc

92794 Issy Moulineaux

Cedex 9

France

Turbo IP Core Getting Started 2

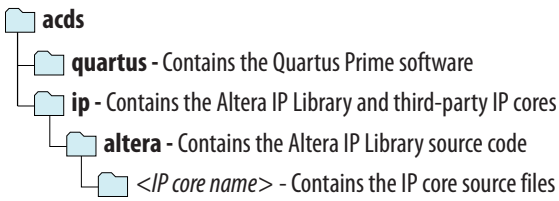
2015.11.11

UG-TURBO

Licensing IP Cores

The Altera IP Library provides many useful IP core functions for your production use without purchasing an additional license. Some Altera MegaCore® IP functions require that you purchase a separate license for production use. However, the OpenCore® feature allows evaluation of any Altera IP core in simulation and compilation in the software. After you are satisfied with functionality and performance, visit the Self Service Licensing Center to obtain a license number for any Altera product.

Figure 2-1: IP Core Installation Path



Note: The default IP installation directory on Windows is ***<drive>*:\altera**<*version number*>; on Linux it is <*home directory*>**/altera/** <*version number*>.

OpenCore Plus IP Evaluation

Altera's free OpenCore Plus feature allows you to evaluate licensed MegaCore IP cores in simulation and hardware before purchase. You need only purchase a license for MegaCore IP cores if you decide to take your design to production. OpenCore Plus supports the following evaluations:

- Simulate the behavior of a licensed IP core in your system.
- Verify the functionality, size, and speed of the IP core quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device with your IP core and verify your design in hardware.

OpenCore Plus evaluation supports the following two operation modes:

- Untethered—run the design containing the licensed IP for a limited time.
- Tethered—run the design containing the licensed IP for a longer time or indefinitely. This requires a connection between your board and the host computer.

Note: All IP cores that use OpenCore Plus time out simultaneously when any IP core in the design times out.

Related Information

- Altera Licensing Site
- Altera Software Installation and Licensing Manual

Turbo IP Core OpenCore Plus Timeout Behavior

All IP cores in a device time out simultaneously when the most restrictive evaluation time is reached. If a design has more than one IP core, the time-out behavior of the other IP cores may mask the time-out behavior of a specific IP core .

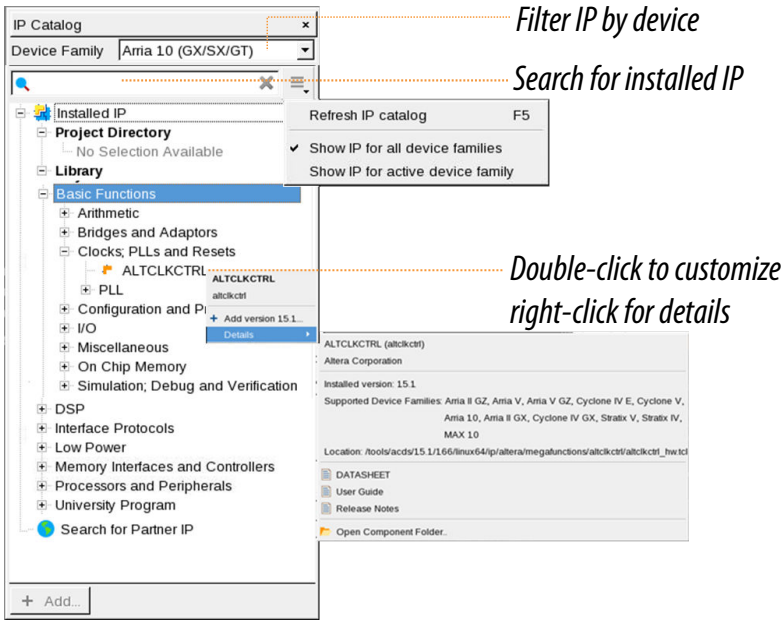
For IP cores, the untethered time-out is 1 hour; the tethered time-out value is indefinite. Your design stops working after the hardware evaluation time expires. The Quartus Prime software uses OpenCore Plus Files (**.ocp**) in your project directory to identify your use of the OpenCore Plus evaluation program. After you activate the feature, do not delete these files..When the evaluation time expires, the data output port `reset_n` goes low, which keeps the IP core permanently in its reset state.

Related Information

- AN 320: OpenCore Plus Evaluation of Megafunctions

IP Catalog and Parameter Editor

The IP Catalog (**Tools** > **IP Catalog**) and parameter editor help you easily customize and integrate IP cores into your project. Use the IP Catalog and parameter editor to select, customize, and generate files representing the custom IP variation in your project.



The IP Catalog displays the installed IP cores available for your design. Double-click any IP core to launch the parameter editor and generate files representing your IP variation. Use the following features to help you quickly locate and select an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**. If you have no project open, select the **Device Family** in IP Catalog.
- Type in the Search field to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, open the IP core's installation folder, and click links to IP documentation.
- Click **Search for Partner IP**, to access partner IP information on the Altera website.

The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Qsys system file (**.qsys**) or IP file (**.qip**) representing the IP core in your project. You can also parameterize an IP variation without an open project.

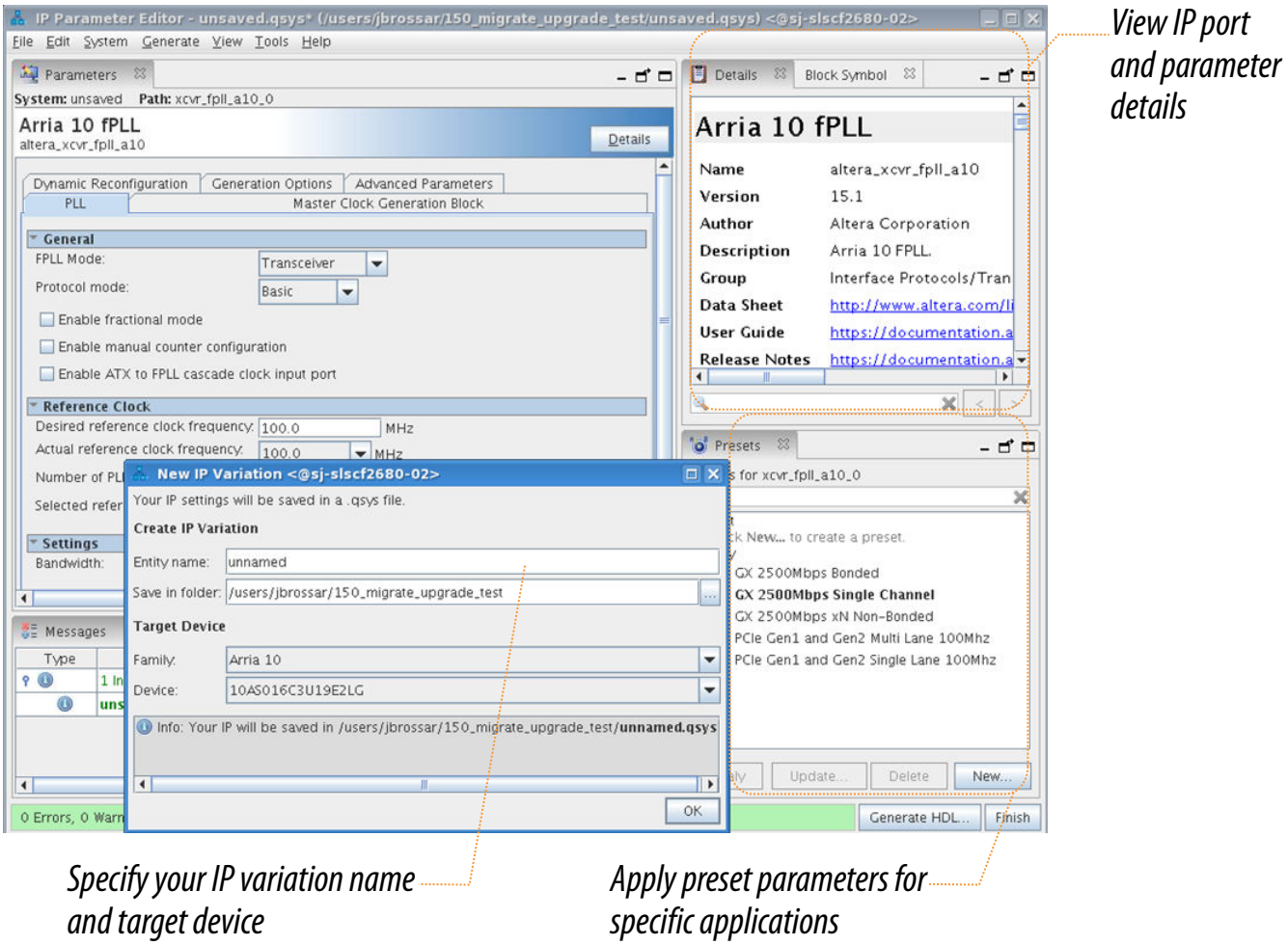
The IP Catalog is also available in Qsys (**View** > **IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the IP Catalog. For more information about using the Qsys IP Catalog, refer to *Creating a System with Qsys* in the *Handbook*.

Note: The IP Catalog (**Tools** > **IP Catalog**) and parameter editor replace the MegaWizard™ Plug-In Manager for IP selection and parameterization, beginning in Quartus II software version 14.0. Use the IP Catalog and parameter editor to locate and paramaterize Altera IP cores.

Generating IP Cores

You can quickly configure a custom IP variation in the parameter editor. Use the following steps to specify IP core options and parameters in the parameter editor.

Figure 2-2: IP Parameter Editor

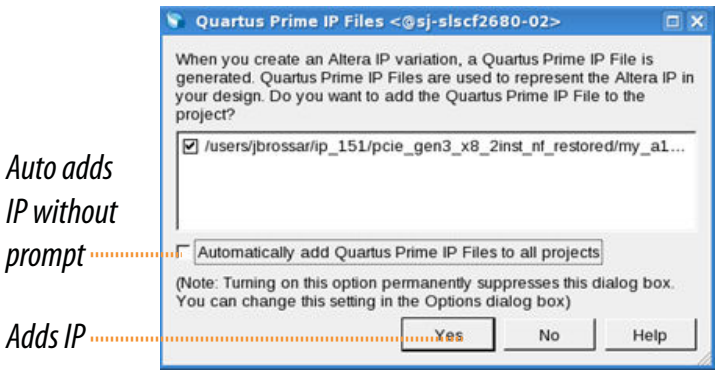


1. In the IP Catalog (**Tools** > **IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name for your custom IP variation. The parameter editor saves the IP variation settings in a file named <*your_ip*>**.qsys**. Click **OK**. Do not include spaces in IP variation names or paths.
3. Specify the parameters and options for your IP variation in the parameter editor, including one or more of the following. Refer to your IP core user guide for information about specific IP core parameters.

- Optionally select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
- Specify parameters defining the IP core functionality, port configurations, and device-specific features.
- Specify options for processing the IP core files in other EDA tools.

4. Click **Generate HDL**. The **Generation** dialog box appears.
5. Specify output file generation options, and then click **Generate**. The IP variation files generate according to your specifications.
6. To generate a simulation testbench, click **Generate** > **Generate Testbench System**.
7. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate** > **HDL Example**.
8. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project. Optionally turn on the option to **Automatically add Quartus Prime IP Files to All Projects**. Click **Project** > **Add/Remove Files in Project** to add IP files at any time.

Figure 2-3: Adding IP Files to Project



For Arria 10 devices and newer, the generated **.qsys** file must be added to your project to represent IP and Qsys systems. For devices released prior to Arria 10 devices, the generated **.qip** and **.sip** files must be added to your project for IP and Qsys systems.

The generated **.qsys** file must be added to your project to represent IP and Qsys systems.

9. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

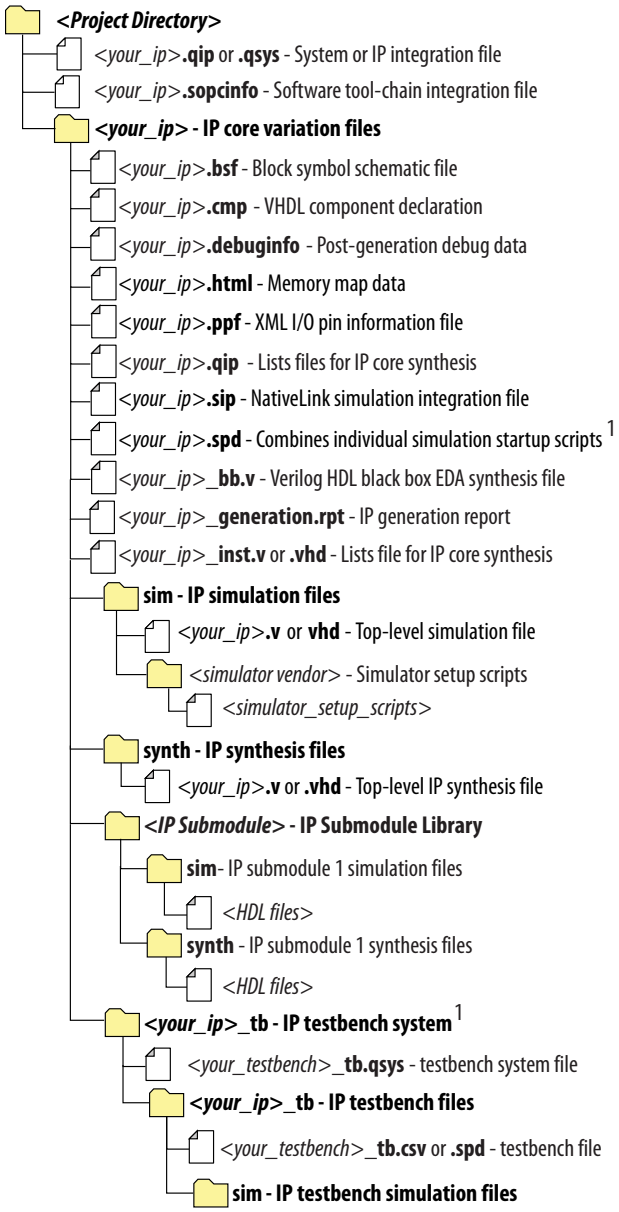
Related Information

- **IP User Guide Documentation**
- **Altera IP Release Notes**

Files Generated for Altera IP Cores and Qsys Systems

The software generates the following output file structure for IP cores and Qsys systems. For Arria 10 devices and newer, the generated **.qsys** file must be added to your project to represent IP and Qsys systems. For devices released prior to Arria 10 devices, the generated **.qip** and **.sip** files must be added to your project to represent IP and Qsys systems. The software generates the following output file structure for IP cores and Qsys systems. The generated **.qsys** file must be added to your project to represent IP and Qsys systems.

Figure 2-4: Files generated for IP cores and Qsys Systems



1. If supported and enabled for your IP core variation.

Table 2-1: IP Core and Qsys Simulation Generated Files

File Name	Description
<*my_ip*>.qsys	The Qsys system or top-level IP variation file. <*my_ip*> is the name that you give your IP variation. You must add the **.qsys** file to your Quartus project to enable NativeLink for Arria 10 and Stratix 10 device families. The Qsys system or top-level IP variation file. <*my_ip*> is the name that you give your IP variation.
<*system*>.sopcinfo	Describes the connections and IP component parameterizations in your Qsys system. You can parse its contents to get requirements when you develop software drivers for IP components. Downstream tools such as the Nios II tool chain use this file. The **.sopcinfo** file and the **system.h** file generated for the Nios II tool chain include address map information for each slave relative to each master that accesses the slave. Different masters may have a different address map to access a particular slave component.
<*my_ip*>.cmp	The VHDL Component Declaration (**.cmp**) file is a text file that contains local generic and port definitions that you can use in VHDL design files.
<*my_ip*>.html	A report that contains connection information, a memory map showing the address of each slave with respect to each master to which it is connected, and parameter assignments.
<*my_ip*>_generation.rpt	IP or Qsys generation log file. A summary of the messages during IP generation.
<*my_ip*>.debuginfo	Contains post-generation information. Used to pass System Console and Bus Analyzer Toolkit information about the Qsys interconnect. The Bus Analysis Toolkit uses this file to identify debug components in the Qsys interconnect.
<*my_ip*>.qip	Contains all the required information about the IP component to integrate and compile the IP component in the software.
<*my_ip*>.csv	Contains information about the upgrade status of the IP component.
<*my_ip*>.bsf	A Block Symbol File (**.bsf**) representation of the IP variation for use in Block Diagram Files (**.bdf**).
<*my_ip*>.spd	Required input file for `ip-make-simscript` to generate simulation scripts for supported simulators. The **.spd** file contains a list of files generated for simulation, along with information about memories that you can initialize.
<*my_ip*>.ppf	The Pin Planner File (**.ppf**) stores the port and node assignments for IP components created for use with the Pin Planner.

File Name	Description
<*my_ip*>_bb.v	You can use the Verilog black-box (**_bb.v**) file as an empty module declaration for use as a black box.
<*my_ip*>.sip	Contains information required for NativeLink simulation of IP components. You must add the **.sip** file to your Quartus project to enable NativeLink for Arria II, Arria V, Cyclone IV, Cyclone V, MAX 10, MAX II, MAX V, Stratix IV, and Stratix V devices.
<*my_ip*>_inst.v or **_inst.vhd**	HDL example instantiation template. You can copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*my_ip*>.regmap	If the IP contains register information, the .**regmap** file generates. The .**regmap** file describes the register map information of master and slave interfaces. This file complements the .**sopcinfo** file by providing more detailed register information about the system. This enables register display views and user customizable statistics in System Console.
<*my_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Qsys system. During synthesis, the .**svd** files for slave interfaces visible to System Console masters are stored in the **.sof** file in the debug section. System Console reads this section, which Qsys can query for register map information. For system slaves, Qsys can access the registers by name.
<*my_ip*>.v or **<*my_ip*>.vhd**	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a ModelSim® script **msim_setup.tcl** to set up and run a simulation.
aldec/	Contains a Riviera-PRO script **rivierapro_setup.tcl** to setup and run a simulation.
/synopsys/vcs **/synopsys/vcsmx**	Contains a shell script **vcs_setup.sh** to set up and run a VCS® simulation. Contains a shell script **vcsmx_setup.sh** and **synopsys_ sim.setup** file to set up and run a VCS MX® simulation.
/cadence	Contains a shell script **ncsim_setup.sh** and other setup files to set up and run an NCSIM simulation.
/submodules	Contains HDL files for the IP core submodule.
<*IP submodule*>/	For each generated IP submodule directory, Qsys generates **/synth** and **/sim** sub-directories.

Simulating Altera IP Cores in other EDA Tools

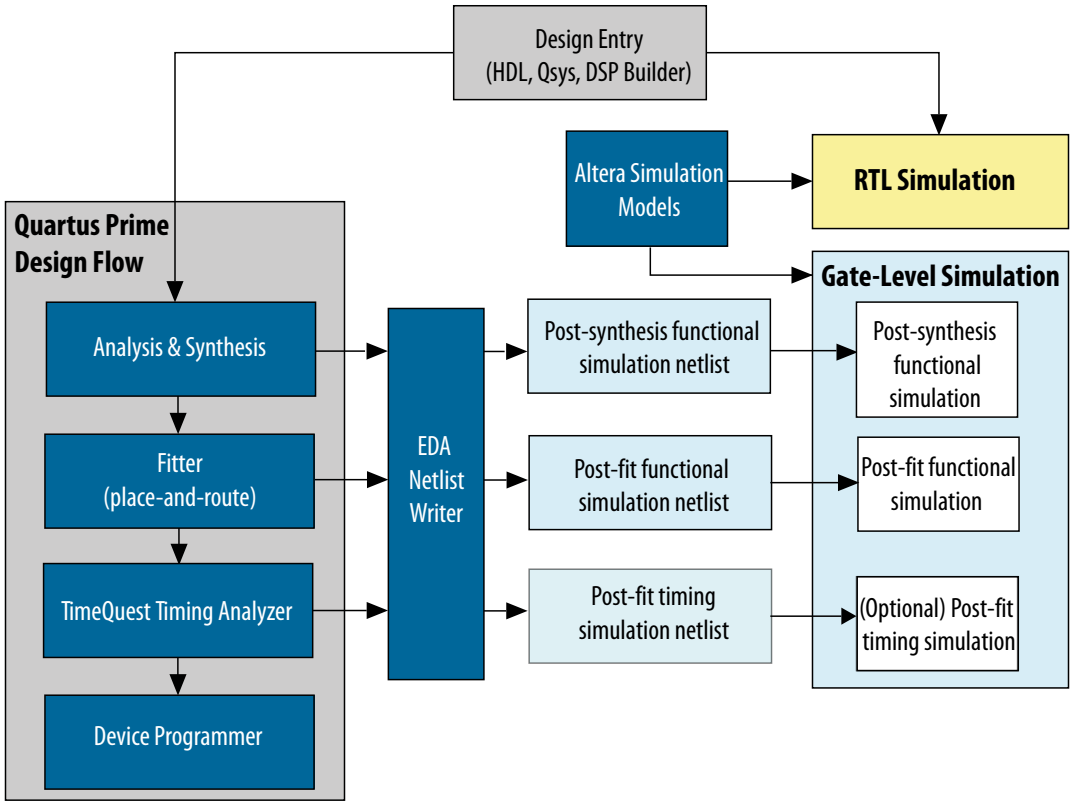
The software supports RTL and gate-level design simulation of Altera IP cores in supported EDA simulators. Simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation.

You can use the functional simulation model and the testbench or example design generated with your IP core for simulation. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench. For a complete list of models or libraries required to simulate your IP core, refer to the scripts generated with the testbench.

You can use the NativeLink feature to automatically generate top-level simulation scripts. NativeLink launches your preferred simulator from within the software. You can use the `ip-setup-simulation` utility to generate a unified, version-agnostic IP simulation script for all Altera IP cores in your design. You can incorporate the IP simulation scripts into your top-level script.

You can use the `ip-setup-simulation` utility to generate a unified, version-agnostic IP simulation script for all Altera IP cores in your design. You can incorporate the IP simulation scripts into your top-level script.

Figure 2-5: Simulation in Design Flow



Note: Post-fit timing simulation is supported only for Stratix IV and Cyclone IV devices in the current version of the software. The Pro Edition software does not support NativeLink RTL simulation. Altera IP supports a variety of simulation models, including simulation-specific IP functional

simulation models and encrypted RTL models, and plain text RTL models. These are all cycle-accurate models. The models support fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some cores, only the plain text RTL model is generated, and you can simulate that model. Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

Note: Altera IP supports a variety of simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. These are all cycle-accurate models. The models support fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some cores, only the plain text RTL model is generated, and you can simulate that model. Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

Related Information

Simulating Altera Designs

DSP Builder Design Flow

DSP Builder shortens digital signal processing (DSP) design cycles by helping you create the hardware representation of a DSP design in an algorithm-friendly development environment.

This IP core supports DSP Builder. Use the DSP Builder flow if you want to create a DSP Builder model that includes an IP core variation; use IP Catalog if you want to create an IP core variation that you can instantiate manually in your design.

Related Information

Using MegaCore Functions chapter in the DSP Builder Handbook.

Turbo IP Core Functional Description

3

2015.11.11

UG-TURBO

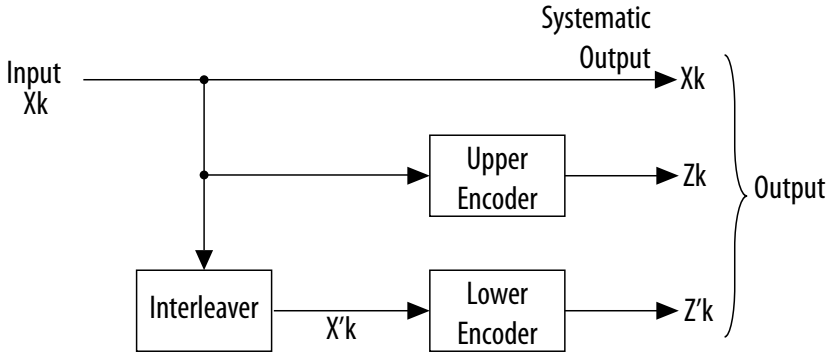
This topic describes the IP core's architecture, interfaces, and signals.

You can parameterize the Turbo IP core as an encoder or a decoder.

Turbo Encoder

The 3GPP Turbo encoder uses a parallel concatenated convolutional code. A convolutional encoder encodes an information sequence and another convolutional encoder encodes an interleaved version of the information sequence. The Turbo encoder has two 8-state constituent encoders and one Turbo code internal interleaver.

Figure 3-1: Turbo Encoder Block Diagram



The output from the turbo coder is:

$X_0, Z_0, Z'_0, X_1, Z_1, Z'_1, ..., X_{K-1}, Z_{K-1}, Z'_{K-1}$

Where:

- Bits $X_0, X_1, ..., X_{K-1}$ are input to both the first 8-state constituent encoder and the internal interleaver (K is the number of bits).
- Bits $Z_0, Z_1, ..., Z_{K-1}$ and $Z'_0, Z'_1, ..., Z'_{K-1}$ are output from the first and second 8-state constituent encoders.
- The bits output from the internal interleaver (and input to the second 8-state constituent encoder) are $X'_0, X'_1, ..., X'_{K-1}$.

Turbo Encoder Data Format

The required input data ordering for a block of size K is: $X_0, X_1, X_2, \ldots . X_{K-1}$. The output data is three bits wide.

Table 3-1: Turbo Encoder Output Data Ordering for a Block of Size *K*

Output Data	source_data		
	2	1	0
0	Z'_0	Z_0	X_0
1	Z'_1	Z_1	X_1
...	...	...	..
$K-1$	Z'_{K-1}	Z_{K-1}	X_{K-1}
K	X_{K+1}	Z_K	X_K
$K+1$	Z_{K+2}	X_{K+2}	Z_{K+1}
$K+2$	X'_{K+1}	Z'_K	X'_K
$K+3$	Z'_{K+2}	X'_{K+2}	Z'_{K+1}

Turbo Encoder Latency Calculation

The encoding delay D is the number of clock cycles the IP core consumes to encode an entire block of data. If K is the block size, $D = K + 14$. The encoding delay does not include the loading delay, which requires the same number of clock cycles as the block size K to load the input data to the input buffer.

For example:

- When $K = 6144$, $D = 6144 + 14 = 6158$
- When $K = 40$, $D = 40 + 14 = 54$

You can calculate the encoding latency (the time taken by the encoder to encode an entire block) using the following equation:

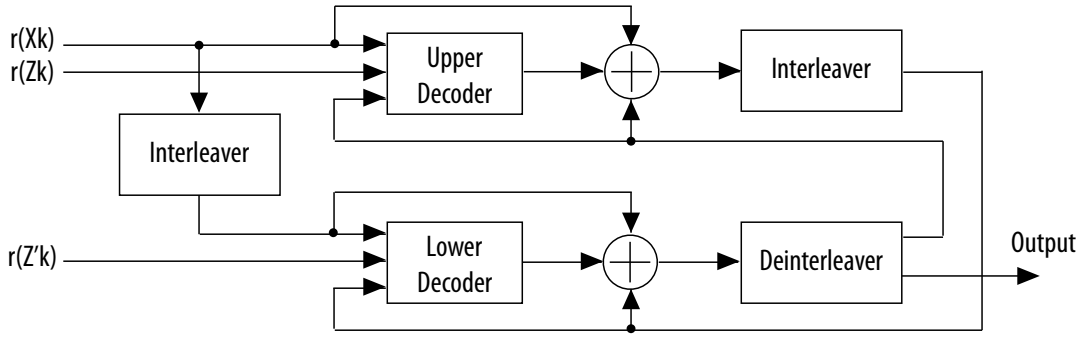
$L = D/f_{MAX}$bps

Where f_{MAX} is the system clock speed..

Turbo Decoder

The Turbo decoder consists of two single soft-in soft-out (SISO) decoders, which work iteratively. The output of the first (upper decoder) feeds into the second to form a Turbo decoding iteration. Interleaver and deinterleaver blocks re-order data in this process.

Figure 3-2: Turbo Decoder Block Diagram



The Turbo decoder supports the MaxLogMAP decoding algorithm. This algorithm is a simplified version of LogMAP that uses less logic resource and offers slightly reduced BER performance relative to LogMAP.

Turbo Decoder Data Format

Input

Table 3-2: Turbo Encoder Output Data Ordering for a Block of Size *K*

Output Data	sink_data		
	3*N* - 1 down to 2*N*	2*N* - 1 down to *N*	*N* - 1 down to 0
0	Z'_0	Z_0	X_0
1	Z'_1	Z_1	X_1
...	...	...	...
K - 1	Z'_{K-1}	Z_{K-1}	X_{K-1}
K	X_{K+1}	Z_K	X_K
K + 1	Z_{K+2}	X_{K+2}	Z_{K+1}
K + 2	X'_{K+1}	Z'_K	X'_K
K + 3	Z'_{K+2}	X'_{K+2}	Z'_{K+1}

The Turbo decoder requires all data to be in the log-likelihood format. The connected system must provide soft information, including parity 1 and parity 2 bit sequences according to the following equation:

$L(x) = \log[P(x=1)/(x=0)]$

The log-likelihood value is the logarithm of the probability that the received bit is a 1, divided by the probability that this bit is a 0. It is represented as a two's complement number. A value of zero indicates equal probability of a 1 and a 0, which you should use for depuncturing. The decoder does not use the most negative two's complement number, which means the representation is balanced.

Table 3-3: Four-bit Mapping Input Values

Input (3 downto 0)	Value
0111	Most likelihood of a 1
...	...
0001	Lowest likelihood of a 1
0000	Equal probability of a 0 or 1
1111	Lowest likelihood of a 0
...	...
1001	Most likelihood of a 0
1000	Not used

Output

The number of output bits can be 1 or 8 bits. For 1 bit, the ordering is: $X_0, X_1, X_2, \ldots . X_{K-1}$

Table 3-4: 8-bit Output Data Ordering

Output Order	source_data							
	7	6	5	4	3	2	1	0
1	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
2	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	X_8
...	...	...	...	...	...	...	...	...
$K/8$	X_{K-1}	X_{K-2}	X_{K-3}	X_{K-4}	X_{K-5}	X_{K-6}	X_{K-7}	X_{K-8}

CRC24A or CRC24B Early Termination

Early termination reduces power consumption and the overall latency, and increases the throughput significantly. It may also increase BER performance of the decoder.

The IP core checks the CRC checksum that the decoder generates after every iteration. Turbo decoding stops as soon as the CRC is successful. turbo decoding does not continue until the maximum number of iterations specified at the input ports. The gains depend on the signal-to-noise ration (SNR) of the received data block, block size, and the maximum number of iterations you specify.

Decoder Latency Calculation

The decoding delay D is the number of clock cycles the IP core consumes to decode an entire block of data.D depends on the block size, the number of iterations to perform, and the number of engines available in the decoder.

The following calculations assume no early termination for the worst case latency.

You can calculate the decoding delay D using one of the following equations:

- If $K < 264$: $D = 26 + (2 \times f(K,N) + 14) \times 2 \times I$
- If $K > 264$, $D = 26 + (f(K,N) + 46) \times 2 \times I$

where:

- *K* is the block size
- *I* is the number of decoding iterations
- *N* is the number of engines specified in the decoder
- f(*K*,*N*) = *K*/*N*, if *K* is divisible by *N*; or f(*K*,*N*) = *K*/8, if *K* is not divisible by *N*

For example:

- D = 26 + (6144/8 + 46) × 2 × 8 = 13,050, if K = 6144, N = 8, I = 8.
- D = 26 + (2 × 40/8 + 14) × 2 × 8 = 410, if K = 40, N = 8, I = 8.

You can calculate the decoding latency (the time the decoder takes to decode an entire block to the decoded data is ready for output) using the following equation:

$L = D/f_{MAX}$ s

Where f_{MAX} is the system clock speed. .

Turbo IP Core Parameters

Table 3-5: Parameters

Parameter	Range	Description
Codec type	Encoder, Decoder	Select an encoder or decoder.
Standard	LTE or UMTS	Select LTE or UMTS
Number of processors	2, 4, 8	Select the number of engines that the decoder uses.
Log-MAP Calculaiton	MaxLogMAP	The decoding algorithm.
Number of input bits	4, 5, 6, 7, 8	The number of input bits to the decoder.
Number of output bits	8	The number of output bits from the decoder.

Turbo IP Core Interfaces and Signals

When designing a datapath that includes a Turbo IP core, you may not need backpressure if you know the downstream components can always receive data. The Turbo IP core Avalon-ST interface has a `READY_LATENCY` value of zero.You may achieve a higher clock rate by driving the `source_ready` signal high, and not connecting the `sink_ready` signal.

Table 3-6: Turbo Encoder Signals

Signal	Direction	Description
clk	Input	Clock signal that clocks all internal registers.

Signal	Direction	Description
reset_n	Input	Active low reset signal. The IP core must always be reset before receiving data. If the megafunction is not reset, the Turbo encoder may produce unexpected results because of feedback signals.
sink_blk_size	Input	Specifies the incoming block size.
sink_data	Input	Input data.
sink_eop	Input	Indicates the end of an incoming packet.
sink_sop	Input	Indicates the start of an incoming packet.
sink_valid	Input	Asserted when data at`sink_data` is valid. When you deassert `sink_valid`, the IP core stops processing until you reassert `sink_valid`.
source_ready	Input	Asserted by the downstream module if it cannot accept data.
sink_error	Input	Error signal indicating Avalon-ST protocol violations on input side. Any non-zero value on `sink_error` causes the Turbo encoder to ignore the current data block. The IP core writes the value that it receives to the `source_error` output port a few cycles later.
sink_ready	Output	Indicates when the IP core can accept data.
source_blk_size	Output	Specifies the outgoing block size.
source_data	Output	Output data.
source_eop	Output	Indicates the end of an outgoing packet.
source_error	Output	Error signal indicating Avalon-ST protocol violations on source side: • 00: No error • 01: Missing start of packet • 10: Missing end of packet • 11: Unexpected end of packet Other types of errors may also be marked as 11.
source_sop	Output	Indicates the start of an outgoing packet.
source_valid	Output	Asserted by the IP core when valid data is available to output.

Table 3-7: Turbo Decoder Signals

Signal	Direction	Description
clk	Input	Clock signal that clocks all internal registers.
reset_n	Input	Active low reset signal. You must always reset the IP core before it receives data. If not reset, the Turbo decoder may produce unexpected results because of feedback signals.

Signal	Direction	Description
CRC_pass	Output	Indicates whether CRC was successful: • 0: Fail • 1: Pass • LTE only.
CRC_type	Output	Indicates the type of CRC that was used for the current data block: • 0: CRC24A • 1: CRC24B • LTE only.
sel_ CRC24A	Input	Specifies the type of CRC that you need for the current data block: • 0: CRC24A • 1: CRC24B • LTE only.
sink_blk_ size	Input	Specifies the incoming block size.
sink_data	Input	Input data.
sink_eop	Input	Indicates the end of an incoming packet.
sink_error	Input	Error signal indicating Avalon-ST protocol violations on input side. Any non-zero value on the `sink_error` port causes the Turbo decoder to ignore the current data block. The IP core writes the value it receives to the `source_error` output port a few cycles later.
sink_max_ iter	Input	Specifies the maximum number of half-iterations.
sink_ready	Output	Indicates when the IP core can accept data.
sink_sop	Input	Indicates the start of an incoming packet.
sink_valid	Input	Assert when data at `sink_data` is valid. When `sink_valid` is not asserted, processing stops until you reassert `sink_valid`.
source_ blk_id	Output	Specifies the outgoing block ID.
source_ blk_size	Output	Specifies the outgoing block size.
source_ data	Output	Output data.
source_eop	Output	Indicates the end of an outgoing packet.

Signal	Direction	Description
source_error	Output	Error signal indicating Avalon-ST protocol violations on source side: • 00: No error • 01: Missing start of packet • 10: Missing end of packet • 11: Unexpected end of packet Other types of errors may also be marked as 11.
source_iter	Output	Shows the number of half iterations after which the Turbo decoder stopps processing the current data block. LTE only.
source_ready	Input	Asserted by the downstream module if it can accept data.
source_sop	Output	Indicates the start of an outgoing packet.
source_valid	Output	Asserted by the IP core when there is valid data to output.

Signals in Qsys Sytems

Qsys systems instantiate all Turbo IP core signals as part of the Avalon-ST data bus.

Table 3-8: Turbo Encoder Data Input

Bits	Signal
13:1	`sink_blk_size`
0	`sink_data`

Table 3-9: Turbo Encoder Data Output

Bits	Signal
15:3	`source_blk_size`
2:0	`source_data`

Table 3-10: Turbo Decoder Data Input

IW is the number of bits for input precision.

Bits	Signal
3*IW+18	`Sel_CRC24A` (LTE only)
3*IW+17:3*IW+13	`sink_max_iter`
3*IW+12 : 3*IW	`sink_blk_size`
3*IW-1:0	`sink_data`

Table 3-11: LTE Turbo Decoder Data Output

Bits	Signal
27	CRC_Pass
26	CRC_type
25:21	source_iter
20:8	source_blk_size
7:0	source_data

Table 3-12: UMTS Turbo Decoder Data Output

Bits	Signal
13:1	source_blk_size
0	source_data

Avalon-ST Interfaces in DSP IP Cores

Avalon-ST interfaces define a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface.

The input interface is an Avalon-ST sink and the output interface is an Avalon-ST source. The Avalon-ST interface supports packet transfers with packets interleaved across multiple channels.

Avalon-ST interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries. Such interfaces typically contain data, ready, and valid signals. Avalon-ST interfaces can also support more complex protocols for burst and packet transfers with packets interleaved across multiple channels. The Avalon-ST interface inherently synchronizes multichannel designs, which allows you to achieve efficient, time-multiplexed implementations without having to implement complex control logic.

Avalon-ST interfaces support backpressure, which is a flow control mechanism where a sink can signal to a source to stop sending data. The sink typically uses backpressure to stop the flow of data when its FIFO buffers are full or when it has congestion on its output.

Related Information

- **Avalon Interface Specifications**

Packet Format Errors

The Turbo IP core has two error signals to communicate data errors in the system:

- `sink_error` is a 2-bit input to receive the up-front error signal.
- `source_error` is a 2-bit output to indicate that there is an error condition (either caught by the Turbo IP core or elsewhere in the previous blocks).

If the IP core receives an error code from `sink_error` during the input of a data block, the IP core assumes the current data block contains some sort of error and discards the data. When the error signal is asserted low, the IP core expects a fresh start-of-packet (`sink_sop` = 1, `sink_valid` = 1) and ignores the data input until it receives a fresh packet.

For misplaced start-of-packet (`sink_sop`) or end-of-packet (`sink_eop`) errors, the IP code indicates an error code depending on the type of error:

- 01 for missing start-of-packet
- 10 for missing end-of-packet
- 11 for unexpected start-of-packet and unexpected end-of-packet

If the LTE standard does not support data block size, the IP core indicates an error signal with the value 11 and ignores the rest of the data block until a fresh start of a packet.

Because of the long processing time of a data block and double-buffering at the inputs and outputs, the IP core reports errors at the input data as soon as they occur. The `source_error` signal might get asserted high at any time during the output of a previous block. When the IP core detects an error, the error code appears for one clock cycle only. If there is more than one error related to a particular data block, the IP core only displays the error code for the first detected error. It takes a few clock cycles to report the detected error at `source_error`.

The IP core may not recover from the error immediately when:

- The detected error is very close to the boundary of the end-of-packet, e.g., when there is a missing end-of-packet, or unexpected start-of-packet or end-of-packet
- A block follows straight after the erroneous block with a different CRC type to the previous block.

.

However, `source_error` indicates errors in all circumstances

Turbo Throughput

You can calculate the throughput using the following equation:

$T = K \text{ x } f_{MAX}/D$ bps

Document Revision History

4

2015.11.11

Turbo IP Core User Guide revision history.

Date	Version	Changes
2015.11.11	15.1	• Corrected performance table M20K and ALM entries. • Updated decoder block diagram • Removed Arria 10 performance table entry
2015.11.01	15.1	Initial release